CHAPTER 4 State-Space Solutions and Realizations

4.3 Computer Computation of CT State-Space Equations

The subject of computer computation of continuous-time differential and integral equations is a vast one. It involves many issues: discretization complexity of an algorithm, efficiency, and numerical stability. The best discretization scheme should yield the most accurate result for a given step size. A good algorithm should be simple to develop, requires a small number of operations (additions and multiplications), and should not be sensitive to parameter variations. These topics are beyond the scope of this text. We discuss in this section only the basic idea and the use of MATLAB functions.

Consider the CT state-space equation

$$\dot{x}(t) = Ax(t) + bu(t)$$

$$y(t) = cx(t) + du(t)$$

Suppose we are asked to compute the output y(t) excited by the input u(*t*) for *t* in $[0, t_f]$ and the initial state x(0). The first step is to select a step size *T* and then discretize the CT state-space equation into a DT state-space equation as

$$y(kT) = c_d x(kT) + d_d u(kT) \tag{4.18}$$

$$x((k+1)T) = A_d x(kT) + b_d u(kT) \tag{4.19}$$

CHAPTER 4 State-Space Solutions and Realizations

If we use the discretization scheme in (4.11), then we have $A_d = I + TA, b_d = Tb$, $c_d = c$, and $d_d = d$. The set of two equations can be computed recursively. We use x(0) and u(0) to compute y(0) from (4.18) and x(T) from (4.19). We then use the computed x(T) and u(T) to compute y(T) and x(2T). Proceeding forward, we can compute y(kT), for k = 0, 1, 2, ...We then carry out interpolation to generate a CT signal y(t) from the DT sequence y(kT). This computation involves only additions and multiplications; it does not encounter the sensitivity issue discussed in Section 4.1. Thus state-space equations are most suited for computer computation.

The remaining question is how to select a step size T. Clearly, the smaller a step size, the more accurate the result. However, it will also require more computation. Thus, in practice, we will not select an unnecessarily small T. The sampling theorem states that if a signal y(t) is bandlimited to ω_{max} and if the sampling period T is selected to be smaller than π / ω_{max}, then y(t) can be recovered exactly from its sampled sequence y(kT).

CHAPTER 4 State-Space Solutions and Realizations

To employ the theorem, we must know the frequency spectrum of y(t), which is generally not available, and use the ideal interpolator, which is not physically realizable. Thus it is not simple to employ the sampling theorem. The easiest way of selecting a T is by trial and error. We select arbitrarily T_1 and $T_2 < T_1$ and carry out the computation. If the two results differ considerably, T_1 is not small enough; whether T_2 is small enough is to be determined by comparing its result with the one computed using a smaller T. We repeat the computation until the results of two subsequent T_i are indistinguishable. If the results of using the original T_1 and T_2 are indistinguishable, T_1 is small enough. In order to find a largest acceptable T, we increase T_1 until its result starts to differ from the preceding computation.

We use an example to illustrate the procedure. Consider the CT state-space equation

CHAPTER 4 State-Space Solutions and Realizations

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & -0.2 \\ 0 & -0.75 & 0.25 \\ 0.5 & -0.5 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} 0.2 \\ 0 \\ 0 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + 0 \times u(t)$$

We use the MATLAB function lsim, abbreviation for *linear simulation*, to compute the output of the system excited by the input u(t) = sin2t for $t \geqslant 0$, and the initial state $x(0) = \begin{bmatrix} 0 & 0.2 & -0.1 \end{bmatrix}'$, where the prime denotes the transpose. We type in an edit window the following:

CHAPTER 4 State-Space Solutions and Realizations

```
%   Program 4.1 ( f41 . m)
a = [ 0 0 -0.2 ; 0 -0.75 0.25 ; 0.5 -0.5 0 ] ; b=[ 0.2 ; 0 ; 0] ;
c = [ 0 0 1 ] ; d= 0 ;
dog = ss ( a , b , c , d) ;
t1 = 0 : 1 : 50 ; u1 = sin ( 2 *t1 ) ; x= [0 ; 0.2 ; -0.1 ] ;
y1 = lsim ( dog , u1 , t1 , x) ;

t2 = 0 : 0.1 : 50 ; u2 = sin ( 2 *t2 ) ;
y2 = lsim ( dog , u2 , t2 , x) ;

t3 = 0 : 0.01 : 50 ; u3 = sin ( 2 *t3 ) ;
y3 = lsim ( dog , u3 , t3 , x) ;
plot ( t1 , y1 , t2 , y2 , ' : ' , t3 , y3 , ' -. ' , [ 0 50 ] , [ 0 0 ])
xlabel ( 'Time (s) ') , ylabel ( 'Amplitude' )
```

CHAPTER 4 State-Space Solutions and Realizations

The first two lines express { A, b, c, d) in MATLAB format. The third line defines the system. We call the system 'dog', which is defined using the state-space model, denoted by ss. The fourth line t1 = 0 : 1 : 50 indicates the time interval [0 ,50] to be computed with increment 1 or, equivalently, with step size $T_1 = 1$ and the corresponding input u1 = sin (2 *t1) . We then use lsim to compute the output y1 = lsim (dog , u1 , t1 , x) . The solid line in Fig. 4.1 is generated by plot (t1 , y1) which also carries out linear interpolation (connecting every pair of neighboring points by a straight line). We then repeat the computation by selecting $T_2 = 0.1$ as its step size and the result y2 is plotted in Fig. 4.1 with a dotted line using plot (t2 , y2 , ' : '). It is quite different from the solid line. Thus the result obtained using $T_1 = 1$ is not acceptable. At this point we don't know whether the result using $T_2 = 0.1$ will be acceptable. We next select $T_3 = 0.01$ and repeat the computation.

CHAPTER 4 State-Space Solutions and Realizations

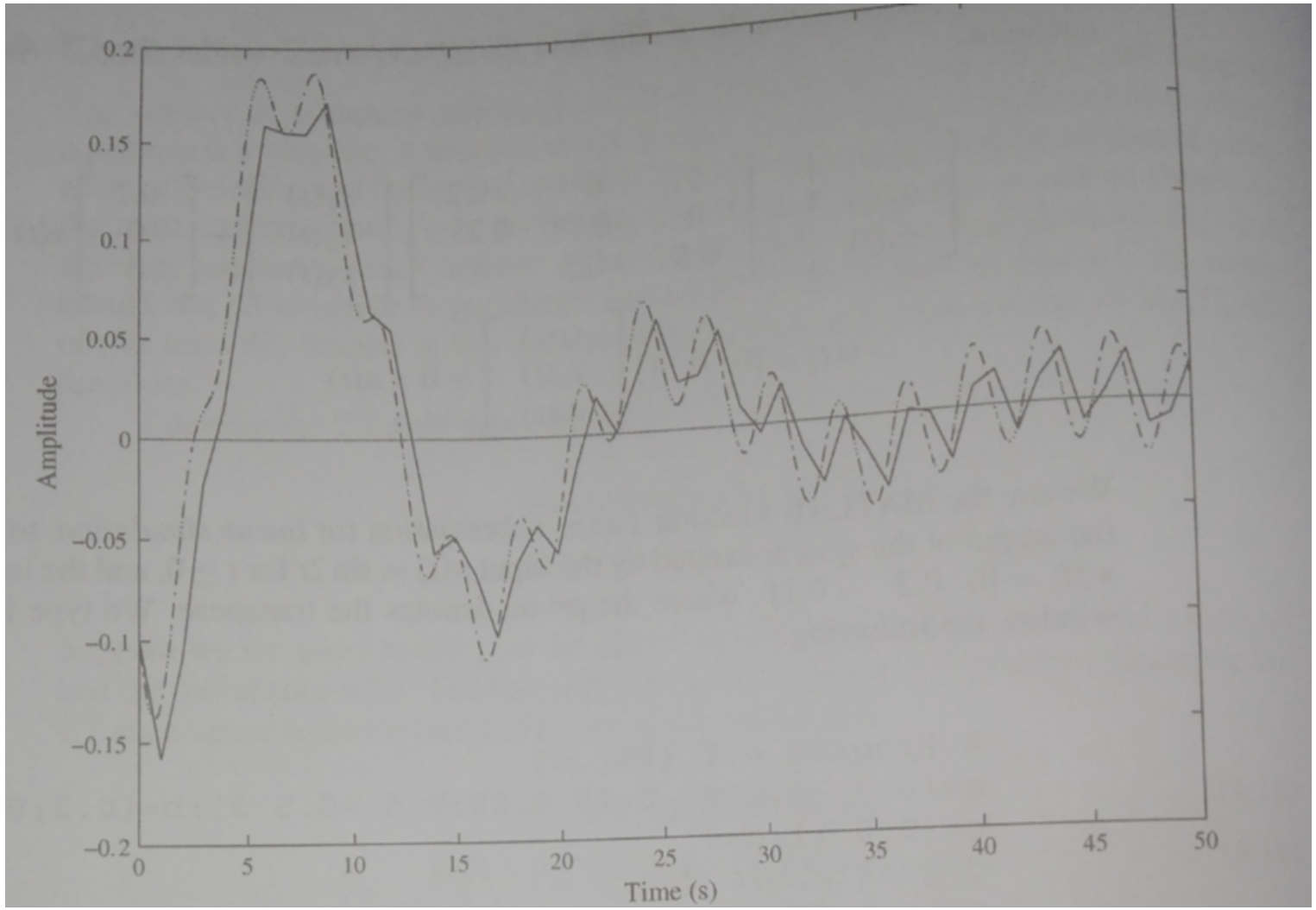

FIGURE 4.1 Outputs computed using $T_1 = 1$ (solid line), using $T_2 = 0.1$ (dotted line), and using $T_3 = 0.01$ (dash-and-dotted line).

CHAPTER 4 State-Space Solutions and Realizations

We then use plot (t3 , y3 , ' -. ') to plot the result in Fig. 4.1 using the dash-and-dotted line. It is indistinguishable from the one using $T_2 = 0.1$. Thus we conclude that the response of the state-space equation can be computed from its discretized equation if the step size is selected to be 0.1 or smaller. Note that the preceding three plot functions can be combined into one as in the second line from the bottom in Program 4.1. The combined plot function also plots the horizontal axis which is generated by plot ([0 50] , [0 0]). We save Program 4.1 as an m-file named f41.m. Typing in the command window f41 will yield Fig. 4.1 in a figure window.

MATLAB contains the function step (a , b , c , d) and impulse （ a , b , c , d ） . The function step computes the output of a system, initially relaxed, excited by a step input and impulse computes the output excited by an impulse input. In using both functions, there is no need to specify the step size and time interval to be computed as in using lsim. Note that both step and impulse are based on lsim. However, they require adaptive selection of step sizes and automatically stop the computation when the responses hardly change.

CHAPTER 4 State-Space Solutions and Realizations

It is not possible to generate an impulse in any computer. Then how is an impulse response generated? Consider

$$\dot{x}(t) = Ax(t) + bu(t)$$

Its integration from t = 0 to t = 0+, where 0+ is infinitesimally larger than 0, with $u(t) = \delta(t)$, yields

$$x(0+) - x(0) = A\int_0^{0+} x(t)dt + b\int_0^{0+} \delta(t)dt = A \cdot 0 + b \cdot 1 = b$$

where the first integration is zero because **x(t)** contains no impulse and the second integration is 1. Thus the impulse input will transfer the initial state from **x(0) = 0** to **x(0+) = b**. Consequently, the impulse response (a zero-state response) can be generated as a zero-input response excited by the initial state **b**. Indeed, this is the way MATLAB generates impulse responses. However, impulse generates a correct response only for a state-space equation with d = 0 or, more precisely, simply ignores $d \neq 0$.

CHAPTER 4 State-Space Solutions and Realizations

MATLAB functions also apply to transfer functions such as step (n , de) and impulse (n , de), where n and de denote the numerator's coefficients and denominator' s coefficients of a transfer function. Internally, MATLAB first uses the function tf2ss to transform a transfer function into a state-space equation and then carries out computation. In Program 4.1, we used the state-space model to define the system as dog=ss(a , b , c , d). We may also define the system as dog=tf (n , de) using the transfer-function model. Recall that transfer functions describe only zero-state responses. Therefore when we use the transfer-function model, the initial state is automatically assumed to be zero. If we use the transfer-function model in Program 4.1, then the result will be different because its initial state is different from zero. If the initial state is zero, there is no difference in using the ss model or tf model.

CHAPTER 4 State-Space Solutions and Realizations

4.3.1 Real-Time Processing

Consider the response denoted by the dash-and-dotted line shown in Fig.4.1. It is the output of the system excited by an input of 50 s long. The elapsed time of computing and plotting the output in Fig.4.1 takes only 0.07s, clocked by the MATLAB functions tic and toc. Thus the time scale in Fig.4.1 has nothing to do with real time. This type of processing is called a non-real-time processing. All computer computation and simulation are non-real-time processing because time and sampling period do not play any role. Only orders of time sequences are important.

Telephone conversation is carried out in real time. SO is audio CD. As soon as a CD player starts to spin, the music starts to appear; it does not wait until the player scans the entire CD. Digital processing of a CT signal always involves a sampling period. For example, the sampling period is T= 1 / 8000 = 0.000125 s or $125\mu s$ for telephone transmission and t = 1 / 44100 = 0.0000226 s or $22.6\mu s$ for audio CD.

CHAPTER 4 State-Space Solutions and Realizations

If the sampling period is larger than the amount of time needed to compute each output y(kT), the specialized digital hardware will store the output in memory and then deliver it at time instant Kt. Because the input u(kT) and the output y(kT) appear at the same real-time instant, it is called a real-time processing. The state-space equation is suited for such a processing. Indeed, consider the state-space equation in (4.18) and (4.19). If it is of dimension 3, then computing each y(kT) requires no more than 20 additions and 20 multiplications. Suppose each addition requires 20 ns (20×10^{-9} s) and each multiplication requires 30 ns, then computing each y(kT) requires at most $1\mu s$ (10^{-6}s). If d = 0, and **x(0) = 0**, then y(0) is automatically 0 and can be delivered at the instant as u(0) arrives. Once u(0) arrives, we can start computing x(T) using (4.19) and then y(T) using (4.18). If the sampling period is $22.6\mu s$ once y(T) is computed (it takes less than $1\mu s$), it is stored in memory. It can be delivered at the same instant as u(T) arrives. Once u(T) arrives, we start to compute x(2T) and then y(2T) and store them in memory. The output y(2T) can be delivered at the instant that u(2T) arrives. Proceeding forward, the output y(kT) can appear at the same instant as u(kT). This is a real-time processing. If $d \neq 0$ an d $x(0) \neq 0$, the delivery of y(kT) must be delayed by one sampling period T because it takes time to multiply u(kT) by d and then to add the product to cx(kT) . Because T is generally very small, y(kT) appears practically at the same instant as u(kT).

CHAPTER 4 State-Space Solutions and Realizations

In conclusion, state-space equations can be used in computer computation and in real-time processing. It can also be used in op-amp circuit implementations as we discuss in the next subsection. Note that transfer functions cannot be used in real-time processing because we can compute the Laplace transform of u(t) only after the entire u(t) for all t in $[0, \infty)$ is available.

4.3.2 Op-Amp Circuit Implementation

Every linear time-invariant (LTI) state-space equation can be implemented using an operational amplifier (op-amp) circuit. Figure 4.2 shows two standard op-amp circuit elements. All inputs are connected, through resistors, to the inverting terminal. Not shown are the grounded noninverting terminal and power supplies. If the feedback branch is a resistor as shown in Fig. 4.2(a), then the output of the element is $-(ax_1 + bx_2 + cx_3)$ If the feedback branch is a capacitor with capacitance C and RC = 1 as shown in Fig. 4.2(b); and if the output is assigned as x, then $\dot{x} = -(av_1 + bv_2 + cv_3)$.

CHAPTER 4 State-Space Solutions and Realizations

We call the first element an adder; the second element, an integrator. Actually, the adder functions also as multipliers and the integrator functions also as multipliers and adder. If we use only one input, say x_1, in Fig. 4.2(a) then the output equals $-ax_1$, and the element can be used as an inverter with gain. Now we use an example to show that every LTI state-space equation can be implemented using the two types of elements in Fig. 4.2.

Consider the state-space equation

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 2 & -0.3 \\ 1 & -8 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} -2 \\ 0 \end{bmatrix} u(t) \tag{4.20}$$

$$y(t) = \begin{bmatrix} -2 & 3 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + 5u(t) \tag{4.21}$$

CHAPTER 4 State-Space Solutions and Realizations

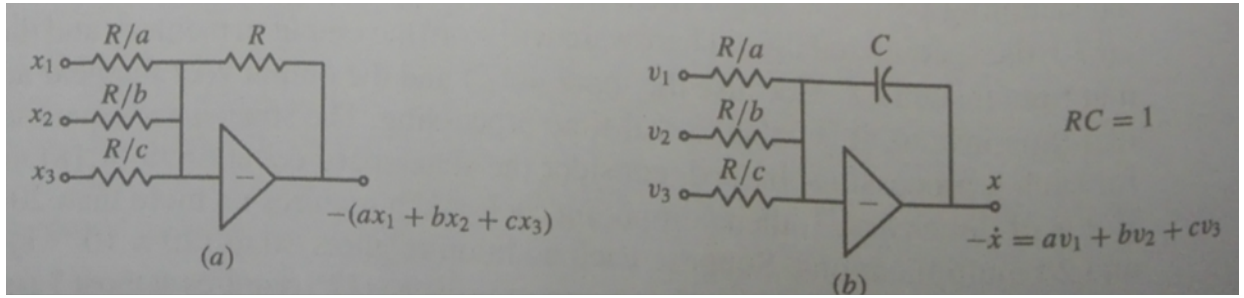

FIGURE 4.2 Two op-amp circuit elements

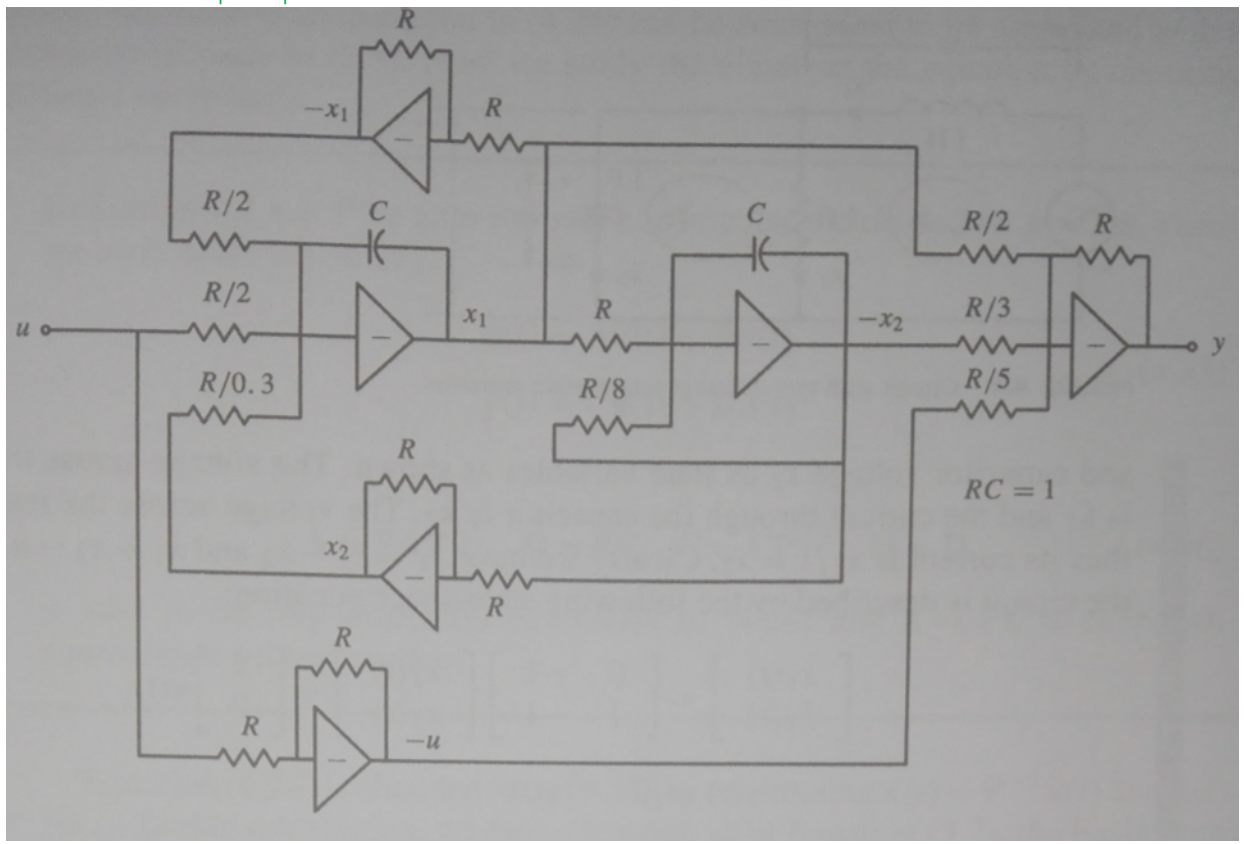

FIGURE 4.3 Op-amp implementation of (4.20) and (4.21)

It has dimension 2 and we need two integrators to implement it. We have the freedom in choosing the output of each integrator as $+x_i$ or $-x_i$. Suppose we assign the output of the left-hand-side (LHS) integrator as x_1 and assign the output of the right-hand-side (RHS) integrator as $-x_2$, as shown in Fig. 4.3. Then the input of the LHS integrator should be, from the first equation of (4.20), $-\dot{x}_1 = -2x_1 + 0.3x_2 + 2u$ and is connected as shown. The input of the RHS integrator should be $\dot{x}_2 = x_1 - 8x_2$ and is connected as shown. If the output of the adder is chosen as y, then its input should equal $-y = 2x_1 - 3x_2 - 5u$ and is connected as shown. Thus the state-space equation in (4.20) and (4.21) can be implemented as shown in Fig. 4.3. Note that there are many ways to implement the same equation. For example, if we assign the outputs of the two integrators in Fig. 4.3 as x_1 and x_2 instead of x_1 and $-x_2$, then we will obtain a different implementation. See References 7 and 10.

In actual operational amplifier circuits, the range of signals is limited by the supplied voltages. If any signal grows outside the range, the circuit will saturate or burn out and the circuit will not behave as the equation does. There is, however, a way to deal with this problem, as we will discuss in a later section.